Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, David Steurer
Kasper Lindberg, Lucas Slot, Hongjie Chen, Manuel Wiedmer

21 October 2024

Algorithms & Data Structures — Exercise sheet 5 — HS 24

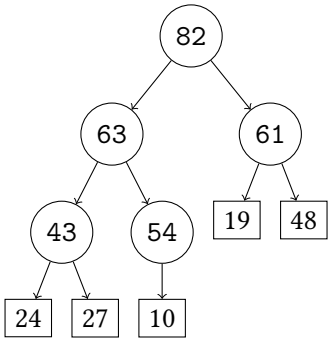
The solutions for this sheet are submitted on Moodle until 27 October 2024, 23:59.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

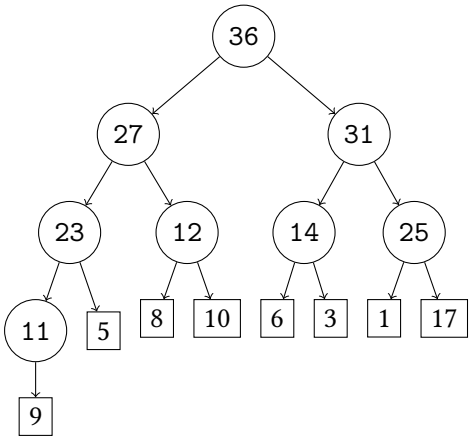
Exercise 5.1 *Max-Heap operations* **(1 point)**.

(a) Consider the following max-heap:



Draw the max-heap after inserting the elements 70 and 51 in that order.

(b) Consider the following max-heap:



Draw the max-heap after two ExtractMax operations.

Exercise 5.2 *Guessing an interval.*

Alice and Bob play the following game:

- Alice selects two integers $1 \leq a < b \leq 200$, which she keeps secret.
- Then, Alice and Bob repeat the following:
  - Bob chooses two integers $0 \leq a' < b' \leq 201$.
  - If $a = a'$ and $b = b'$, Bob wins.
  - If $a' < a$ and $b < b'$, Alice tells Bob 'my numbers are strictly between your numbers!'.
  - Otherwise, Alice does not give any clue to Bob.

Bob claims that he has a strategy to win this game in 12 attempts at most. Prove that such a strategy cannot exist.

Hint: *Represent Bob's strategy as a decision tree. Each edge of the decision tree corresponds to one of Alice's answers, while each leaf corresponds to a win for Bob.*

Hint: *After defining the decision tree, you can consider the sequence $k_0 = 1$ and $k_n = 2k_{n-1} + 2$ for $n \geq 1$, and prove that $k_n = 3 \cdot 2^n - 2$ for any $n \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$. The number of vertices in the decision tree should be related to k_n.*

Exercise 5.3 *Quick(?) sort* **(1 point)**.

Recall the pseudocode for the *quick sort* algorithm from the lecture:

Algorithm 1 quick sort

```
1:  function QuickSort(A, ℓ, r)
2:      if ℓ < r then
3:          k = Partition(A, ℓ, r)
4:          QuickSort(A, ℓ, k − 1)
5:          QuickSort(A, k + 1, r)
6:  function Partition(A, ℓ, r)
7:      i ← ℓ
8:      j ← r − 1
9:      p ← A[r]                          ▷ Choose the rightmost entry as pivot
10:     repeat
11:         while i < r and A[i] ≤ p do
12:             i ← i + 1
13:         while j ≥ ℓ and A[j] > p do
14:             j ← j − 1
15:         if i < j then
16:             Swap A[i] and A[j]
17:     until i > j
18:     Swap A[i] and A[r]                ▷ At the end, the correct place for the pivot is i
19:     Return i
```

We want to study the number of comparisons between array entries the quick sort algorithm performs when we apply it to an array $A[1 \ldots n]$ consisting of n unique integers which is already sorted in ascending order (so $A[1] < A[2] < \ldots < A[n]$).

(a) Show that the number of comparisons $T(n)$ between array entries that QuickSort$(A, 1, n)$ performs when applied to a sorted array A as above, and with the above rule to select the pivot satisfies the recursive relation

$$T(1) = 0, \quad T(n) = T(n-1) + (n-1) \quad \forall n \geq 2.$$

You may assume for simplicity that Partition(A, ℓ, r) always performs exactly $r - \ell$ comparisons between entries. In your argument, refer to the pseudocode above.

(b) Assume $n \geq 3$. Show that $T(n) = \Theta(n^2)$. To do so, first give an exact expression for $T(n)$ based on the recursive formula of part (a) (your exact expression does not need to be maximally simplified, e.g., it is allowed to contain summation-symbols).

Hint: *Based on the recursive formula from part (a), how could you write $T(n)$ in terms of $T(n-2)$? How could you write it in terms of $T(n-3)$? Repeat this process.*

Exercise 5.4 *Building a Heap* **(1 point)**.

Recall that a binary tree is called *complete* if all of its layers are fully filled, except possibly the last layer, which should be filled from left to right. A *(max-)heap* is a complete binary tree with the extra property that for any node C with parent P,

$$\text{key}(P) \geq \text{key}(C). \qquad \text{(heap-condition)}$$

Also recall that for a tree T, the root is at level 0 and the leaves are at level $\text{height}(T)$; for a node at level ℓ, its children are at level $\ell + 1$.

In this exercise, we formally prove the correctness of the following algorithm from lecture, which converts any complete binary tree into a heap.

Algorithm 2 Heap Construction

```
function Heapify(T)
    for t = height(T) − 1, ..., 0 do
        for nodes N at level t do
            for ℓ = t, ..., height(T) − 1 do
                C1 ← the left child of N, if no such child exists assign it key −∞.
                C2 ← the right child of N, if no such child exists assign it key −∞.
                if key(C1) ≥ key(C2) and key(C1) > key(N) then
                    Swap the keys of nodes N and C1.
                    N ← C1
                else if key(C1) < key(C2) and key(C2) > key(N) then
                    Swap the keys of nodes N and C2.
                    N ← C2
                else
                    Exit inner for loop
```

Let T be a complete binary tree consisting of n nodes with $n \geq 2$. Let H be the data structure that results from executing Heapify(T).

(a) Prove that the executing Heapify(T) returns a valid heap.

Hint: *Use the invariant $I(t)$ for $0 \leq t \leq \text{height}(T)$: all nodes from levels $\text{height}(T), \ldots, t$ satisfy the heap condition, namely $\text{key}(P) \geq \text{key}(C)$ where P is the parent node of level at least t, and C is a child of P.*

(b)* Prove that the runtime of executing $\text{Heapify}(T)$ takes time $O(n)$.

You may use the fact that for any $k \in \mathbb{N}$

$$\sum_{i=1}^{k} \frac{i}{2^i} \leq 2\,.$$

Hint: *Write the runtime as an outer sum over the various levels and an inner sum over all the nodes of that level.*

Data structures.

Exercise 5.5 *Implementing abstract data types.*

In the lecture, you saw how we can implement the abstract data type list with operations insert, get, delete and insertAfter. In this exercise, the goal is to see how we can implement two other abstract data types, namely the stack (german "Stapel") and the queue (german "Schlange" or "Warteschlange"). The abstract data type stack is, as the name suggests, a stack of elements. For a stack S, we want to implement the two following operations; see also Figure 1.

- $\text{push}(x, S)$: Add x on top of the stack S.
- $\text{pop}(S)$: Remove (and return) the top element of the stack S.

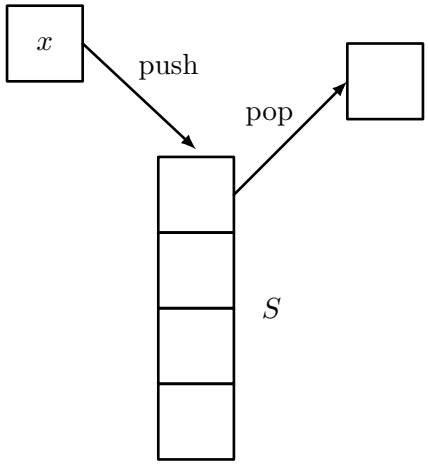


Figure 1: Abstract data type stack

The abstract data type queue is a queue of elements. For a queue Q, we want to implement the following two operations; see also Figure 2.

- $\text{enqueue}(x, Q)$: Add x to the end of Q.
- $\text{dequeue}(Q)$: Remove (and return) the first element of Q.

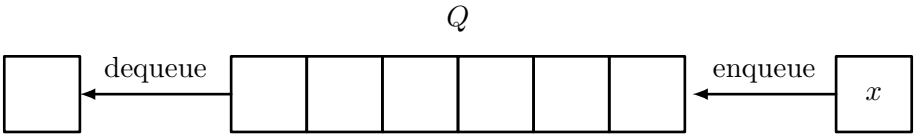


Figure 2: Abstract data type queue

(a) Which data structure from the lecture can be used to implement the abstract data type stack efficiently? Describe for the operations push and pop how they would be implemented with this data structure and what the run time would be.

(b) Which data structure from the lecture can be used to implement the abstract data type queue efficiently? Describe for the operations enqueue and dequeue how they would be implemented with this data structure and what the run time would be.